Master's Thesis in Graduate School of Library, Information and Media Studies

SNN Meets ANN: Combining Spiking Neural Network (SNN) and Artificial Neural Network (ANN) for Image Classification

March 2021

201821636

Muramatsu Naoya

SNN Meets ANN: Combining Spiking Neural Network (SNN) and Artificial Neural Network (ANN) for Image Classification

SNN・ミーツ・ANN
―スパイキングニューラルネットワークと人工ニューラルネットワークの組み合わせによる画像分類―

Student No.: 201821636
氏名: 村松　直哉
Name: Muramatsu Naoya

In recent years, deep learning has achieved breakthrough successes in various fields, such as image recognition, natural language processing, playing perfect-information game and so on. Unfortunately, the traditional deep neural networks, referred to as artificial neural networks (ANNs), suffer from high energy consumption. Therefore, spiking neural networks (SNNs) have been proposed, which can dramatically decrease the computational power during training and inference of machine learning models. However, SNNs are inferior to conventional neural networks in terms of performance even for the simplest task, image classification. The reason is that data handled in the real world often have been normalized into the range of $[0, 1]$, which is not appropriate for SNNs processing spikes, sparse and binary signals.

To cope with the aforementioned issues, we appeal to combine SNN and ANN, and propose hybrid neural networks (HNNs). An HNN consists of spiking layers (SLs, i.e., the neural layers of an SNN) and artificial layers (ALs, i.e., the neural layers of an ANN). Thanks to the coding methods, such as the existing ones (duplicate coding and Poisson coding) and our proposed differentiable Gaussian coding, we can bridge the two kinds of neural layers by converting the real-valued data into binary spike trains. Furthermore, we introduce two learning methods, which enables us to train SLs and ALs either separately or simultaneously.

To demonstrate the effectiveness of the proposed framework, we conducted a series of experiments based on the widely used datasets, MNIST and CIFAR-10. The experimental results show that: (1) The performance of a network can generally be improved by increasing the ratio of ALs, which can surpass that of pure SNNs. (2) Gaussian coding is capable of achieving higher accuracy for complex datasets even when the percentage of ALs is small. The proposed HNNs can bridge the gap between the traits of ANNs and SNNs, which provides insights on well understanding the potential of SNN.

Principal Academic Advisor: Tetsuji SATOH
Secondary Academic Advisor: Hai-Tao YU

SNN Meets ANN: Combining Spiking Neural Network (SNN) and Artificial Neural Network (ANN) for Image Classification

Muramatsu Naoya

Graduate School of Library, Information and Media Studies

University of Tsukuba

March 2021

Contents

List of Figures

Chapter 1

Introduction

Sophisticated neural networks have great potential to take over more and more of the tasks that were previously done solely by biological intelligence, in all aspects of life and industry. As the core and foundation of computer science, the realization of good machine learning models with low power consumption has always been the most challenging task. Turning to the biological world, high intelligence and energy efficiency are essential for all components, and are deeply rooted in the evolutionary process of both the nervous system. Therefore, the problem of artificial intelligence can be thought of as taking inspiration from living things and applying their operating principles to mimic similar capabilities.

Real world problems are so complex that simple rule-based methods e.g., template matching in image processing, cannot deal with all of them with high accuracy. Artificial neural networks (ANNs) have emerged to solve this problem, and are currently achieving amazing intelligence in natural language processing [1–3], image analysis and recognition [4–7], 3D reconstruction from an image [8], and complex mind sports [9]. However, it basically relies on huge computational resources, time, and power to learn a few specific tasks, which leads to large latency [10]. For example, in recent machine learning models, GPUs such as the NVIDIA Tesla V100 and A100 (250 W) are used to solve a single task [11] **TODO:** *other examples*, while the human brain requires only about 20 W [12] and can handle any task related to life activities. In addition, the human brain can operate for about a hundred years without major updates e.g., reconfigurations of its overall architecture. Consequently, the current widespread use of ANNs is a considerable disadvantage, especially in mobile applications where real-time response is critical and energy supply is limited.

To overcome this gap between biological brains and ANNs, spiking neural networks (SNNs), which have behavioral mechanisms more similar to biological neurons, have been attracting attention. In most ANNs, information is processed between neurons in the form of continuous values, the range of $[0, 1]$ in most cases, but in SNNs, information is represented and processed by spikes, which is binary sparse data streaming temporally. It is believed that animals use these intermittent signals to reduce their overall energy consumption and recognize the spatio-temporal data coming from these sensory organs. By mimicking this mechanism, it is hoped not only that low-power machine learning models can be realized, but also that spatio-temporal data can be processed in real time [13].

Sensor data acquired from the real world is often represented as real-continuous values, and in order to process them, many datasets and machine learning systems are created assuming continuous value data. However, since SNNs depict data as spike signals, they are

generally not suitable for handling continuous-valued data. In addition, widely available pretrained models that assume ANNs cannot be used for SNNs directly. In this thesis, we proposed hybrid neural network (HNN) that utilizes the advantages of both ANN and SNN, as well as the paradigm for its construction and training methods, and validate them with image classification tasks.

In the following sections, we described the details of our thesis from two aspects: why we conducted this research (Section 1.1) and what is contributions we achieved (Section 1.2). In addition, the outline of this thesis is shown in the last section (Section 1.3).

1.1 Motivation

SNNs have great engineering potential for energy-efficiency and low-latency and neuroscientific curiosity for the more brain-like mechanism; each neuron uses impulses and spikes to process and communicate information. It is important to note that while this is a more similar method to biological neurons, it is not exactly the same. A biological neuron is not a simple spike generator using a delta function, and different types of neurons have different shapes of the spiking signals they generate. The SNN is based on the assumption that the phenomenon of spike generation itself plays the most important role in the information representation of biological neurons. While the computation of traditional ANNs can be understood as an approximation of the temporal impulse rate, event-driven SNNs perform computations based on individual spikes with precise timing considerations. That is why SNNs are introduced as the third generation of ANNs [14].

However, the current performance of SNNs does not exceed that of ANNs, and the hardware suitable for SNNs is not widely available. Therefore, despite the rapid development of SNNs in the last decade, they have not become a replacement for traditional ANNs. One of the key problems in SNNs is the poor performance on ANN-oriented datasets in which the data have the continuous-values format. With the development of backpropagation for SNNs, SNNs can match or exceed the performance of ANNs on input data represented by spiking signals such as N-MNIST [15] and DVS128 gesture [16], but their performance on the widespread ANN-oriented datasets is not as high as ANNs. In addition, ANNs have achieved outstanding results in diverse tasks, and trained models on large datasets are utilitarian in research and product development. However, SNNs cannot directly use these trained ANNs, and they need to be trained from scratch.

In order to solve these problems, we believe it is important to build a model that takes advantage of both ANN and SNN. By establishing this fused framework, we can effectively use the public resources built on the ANNs and contribute to the further development of ANNs and SNNs. Furthermore, this can facilitate the transition from ANNs to SNNs.

1.2 Contributions

For overcoming the challenges of SNNs detailed in the previous section, this research has made contributions in three main ways. The most important one is that we proposed the novel neural network architecture, called Hybrid Neural Network (HNN). Traditionally, the typical method of associating ANNs and SNNs is to convert a trained ANN into an SNN (discussed in Section 3.1.1). This conversion approach was proposed as a way to overcome

the low learning performance of SNNs due to the unavailability of backpropagation, and many effective methods have been proposed [17–20]. However, in HNN, an ANN and an SNN are directly connected, which is completely different from the converting methods. Currently, the learning performance of SNNs is lower than that of ANNs, and various methods are still under discussion. In this context, proposing a new method to improve the performance of SNNs is important for future SNN research because it can increase the number of available options.

Secondly, we proposed elemental implementations for constructing HNN. In particular, we implemented and evaluated three coding methods, including differentiable ones. As with conventional SNNs dealing with continuous-valued input, HNNs require coding process to convert real-valued data into spike trains. Although the coding process itself does not need to be differentiable in ordinary SNNs, HNNs require differentiable coding in order to learn artificial layers (ALs) composed of artificial neurons in ANN and spiking layers (SLs) composed of spiking neurons in SNN, simultaneously.

Third, we explored the properties of components of the HNN. Each component of HNNs (i.e., ALs, SLs and coding) has various choices, which gives researchers and developers a wide range of options, and yet it is necessary to know the characteristics of each of them in detail in order to improve the performance of the entire network. In particular, the composition of ALs and SLs of an HNN is one of the most important factors. Generally speaking, as the ratio of AL increased, the performance of HNN improved and recorded higher accuracy than pure SNNs as the baselines. Two learning methods for HNNs were also presented and their properties were evaluated. The results show that we can improve the performance by learning ALs and SLs separately, and in addition, we can handle more complex problems by learning ALs and SLs simultaneously.

1.3 Thesis Outline

This thesis consists of six chapters, detailing the background knowledge of SNNs, illustrating the concept and individual components of HNNs, and evaluating their performance by one of the most basic tasks, image classification. The theoretical background of SNN is described in Chapter 2, focusing on the knowledge utilized in the HNN. The components of HNNs are detailed in Chapter 4, which reveals the implementation method and important hyperparameters of HNNs. The experimental conditions and the results for evaluating HNNs are described in Chapter 5. We summarize the entire thesis and provide future guidelines for the HNN research in Chapter 6.

Chapter 2

Spiking Neural Network

In this chapter, we give an explanation of the theoretical background of Spiking Neural Networks (SNNs). We describe the components of a single neuron on a SNN (Section 2.1) and the formulated neuron model we utilized (Section 2.2). In Section 2.3, the approximation of the spike-generating function is introduced, which is necessary for applying backpropagation to SNNs.

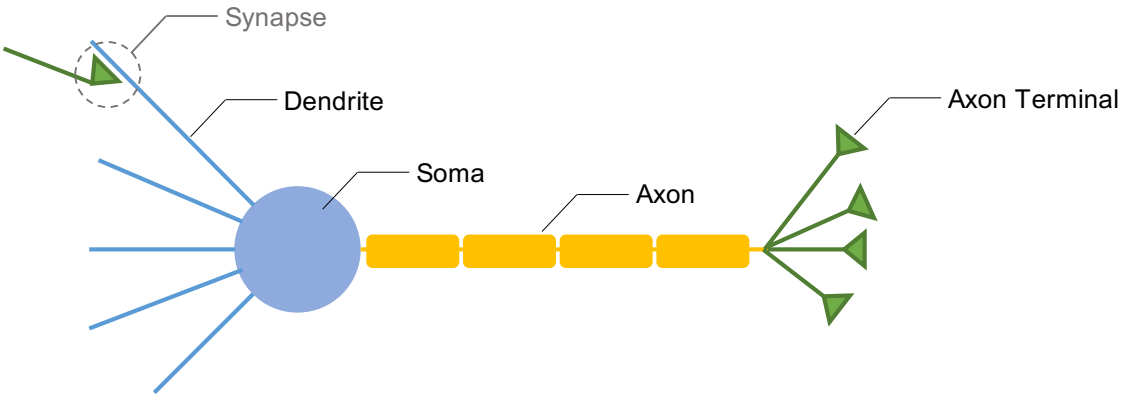
2.1 The Structure of a Neuron

As shown in Figure 2.1(a), a biological neuron changes its membrane potential, internal voltage, in the soma in response to an input signal through the synapses. When the membrane potential exceeds the threshold voltage, a spike signal is generated from the soma and transmitted to the next neuron through the axon and axon terminal.

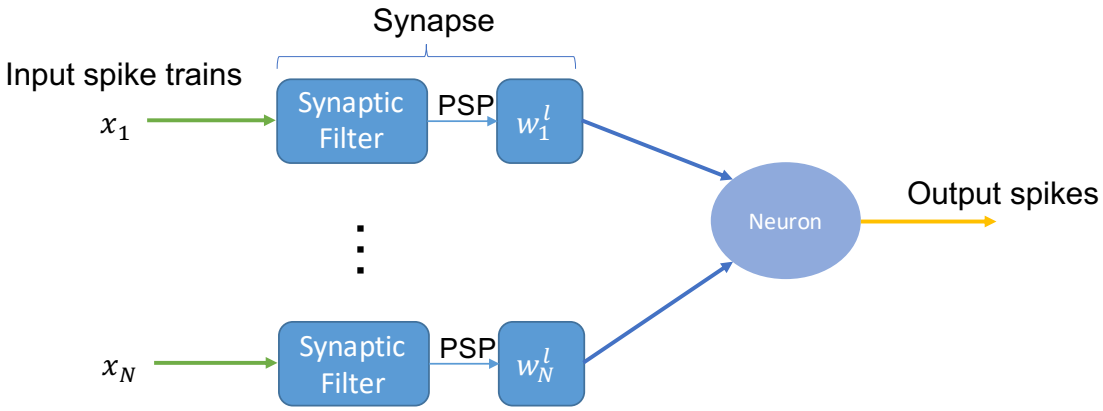
As shown Figure 2.1(b), in many SNN studies synapses and neurons are treated as separate modules. Therefore, synapses receive spike signals from the pre-synaptic neurons and generate postsynaptic potential (PSP). The PSP is weighted for each synaptic and flows into the postsynaptic neuron (biologically this is the input current to a neuron). The membrane potential of each neuron varies with the input current (weighted PSP) in time. When it exceeds the threshold voltage, an output spike is generated from that neuron, and immediately afterward, its membrane potential plummets to the rest voltage. As each neuron repeats the above temporal behavior, information flows on spike signals. Because the current flows only when a spike is generated, SNN can achieve lower power consumption.

2.2 Neuron and Synapse Model

In this thesis, we used a model of neurons and synapses formulated by infinite impulse response (IIR) filters [21]. As a general form in the discrete-time domain, SNNs can be



(a) A biological neuron.



(b) A spiking neuron model as IIR filters.

Figure 2.1: Comparison between a biological neuron and a spiking neuron model.

interpreted as a network of IIR filters:

$$V_i^l[t] = \lambda V_i^l[t-1] + I_i^l[t] - V_{th} R_i^l[t] \tag{2.1a}$$

$$I_i^l[t] = \sum_j^{N_{l-1}} w_{i,j}^l F_j^l[t] \tag{2.1b}$$

$$R_i^l[t] = \theta R_i^l[t-1] + O_i^l[t-1] \tag{2.1c}$$

$$F_j^l[t] = \sum_{p=1}^{P} \alpha_{j,p}^l F_j^l[t-p] + \sum_{q=0}^{Q} \beta_{j,q}^l O_j^{l-1}[t-q] \tag{2.1d}$$

$$O_i^l[t] = U(V_i^l[t] - V_{th}) \tag{2.1e}$$

$$U(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases} \tag{2.1f}$$

where l and i denote the index of layer and neuron respectively, and j denotes input index and t is the time step, N_l is a number of neurons in the l-th layer. $V_i^l[t]$ is neuron membrane potential, and V_{th} is the threshold (a neuron fires when its membrane potential overcomes it). $I_i^l[t]$ is weighted input (weighted PSP). $R_i^l[t]$ is reset voltage to decrease membrane potential to rest voltage after the neuron firing, $F_j^l[t]$ is PSP. $O_i^l[t]$ is a spike function to describe the conditions of firing, and $U(x)$ is a Heaviside step function. P and Q denote the feedback and feedforward orders. $\lambda, \theta, \alpha_{j,p}^l$ and $\beta_{j,q}^l$ are coefficients of neuron filter, reset filter and synapse filter respectively. To change these coefficients, this model of neuron can represent various neuron models.

To simplify, all the neurons in our experiments behaved as leaky integrate-and-fire (LIF) neurons [22], which is the most basic, widely used type of phenomenological neuron model [23], with the following setting:

$$\alpha_p = 0, p \in \{1, 2, \cdots, P\} \tag{2.2a}$$

$$\beta_0 = 1 \tag{2.2b}$$

$$\beta_q = 0, q \in \{1, 2, \cdots, Q\} \tag{2.2c}$$

For these settings, Equation 2.1d can be simplified to $F_j^l[t] = \beta_{j,0}^l O_j^{l-1}[t]$ as in previous works [19, 24].

2.3 Approximated Backpropagation

As this thesis focuses on image classification tasks, we utilized cross-entropy loss E with probability each neuron in the last layer firing in a fixed-length time window T:

$$E = -\sum_i^{N_L} y_i \ln p_i \tag{2.3a}$$

$$p_i = \frac{\exp(\sum_t^T O_i^L[t])}{\sum_{j=1}^{N_L} \exp(\sum_t^T O_j^L[t])} \tag{2.3b}$$

where y_i is a one-hot vector representing the truth label, L is the number of layers of the SNN, $O_i^L[t]$ denotes the output of the last layer.

Through chain rule, the gradient can be represented:

$$\frac{\partial E}{\partial w^l} = \sum_{t=1}^{T} \boldsymbol{\delta}^l[t]\boldsymbol{\epsilon}^l[t] \left(\boldsymbol{F}^l[t] + \sum_{i=1}^{t-1} \boldsymbol{F}^l[i] \prod_{j=i}^{t-1} \boldsymbol{\kappa}^l[j] \right) \tag{2.4}$$

where

$$\begin{aligned} \delta^l[t] &= \frac{\partial E}{\partial O_i^l[t]} \\ &= \sum_{q=0}^{Q} \sum_{j}^{N_{l+1}} \frac{\partial E}{\partial O_j^{l+1}[t+q]} \frac{\partial O_j^{l+1}[t+q]}{\partial O_i^l[t]} + \frac{\partial E}{\partial O_i^l[t+1]} \frac{\partial O_i^l[t+1]}{\partial O_i^l[t]} \end{aligned} \tag{2.5a}$$

$$\begin{aligned} \kappa_i^l[t] &= \frac{\partial V_i^l[t+1]}{\partial V_i^l[t]} \\ &= \lambda - V_{th}\epsilon_i^l[t] \end{aligned} \tag{2.5b}$$

$$\epsilon_i^l[t] = \frac{\partial U(V_i^l[t] - V_{th})}{\partial V_i^l[t]} \tag{2.5c}$$

The gradients between each layer can be calculated:

$$\frac{\partial O_i^l[t+1]}{\partial O_i^l[t]} = \frac{\partial O_i^l[t+1]}{\partial V_i^l[t+1]} \frac{\partial V_i^l[t+1]}{\partial R_i^l[t+1]} \frac{\partial R_i^l[t+1]}{\partial O_i^l[t]} \tag{2.6a}$$

$$= -V_{th}\delta_i^l[t+1]\epsilon_i^l[t+1] \tag{2.6b}$$

$$\frac{\partial O_j^{l+1}[t+q]}{\partial O_i^l[t]} = \frac{\partial O_j^{l+1}[t+q]}{\partial V_j^{l+1}[t+q]} \frac{\partial V_j^{l+1}[t+q]}{\partial I_j^{l+1}[t+q]} \frac{\partial I_j^l[t+q]}{\partial O_i^l[t]} \tag{2.6c}$$

$$= \beta_{j,q}^{l+1}\delta_j^{l+1}[t+q]\epsilon_j^{l+1}[t+q]w_{j,i}^{l+1} \tag{2.6d}$$

Here $U(x)$ is not differentiable, which is the main obstacle for SNN learning algorithms. Fang *et al.* assumed that when Gaussian noise $\mathcal{N}(0,\, \sigma^2)$ is nipped in, LIF neurons can be approximated by Poisson neurons such that the firing rate follows [21]:

$$P(v) = \frac{1}{2}\mathrm{erfc}(\frac{V_{th} - v}{\sqrt{2}\sigma}) \tag{2.7}$$

where $\mathrm{erfc}(\cdot)$ represents a complementary error function. Thus, the derivative of $U(x)$ can be approximated [25]:

$$\frac{\partial U(x)}{\partial v} \approx \frac{\partial P(v)}{\partial v} = \frac{e^{-\frac{(V_{th}-v)^2}{2\sigma^2}}}{\sqrt{2\pi}\sigma} \tag{2.8}$$

Chapter 3

Related Work

Unlike ANNs, the data are represented as spike trains in SNNs, which has the advantages and disadvantages. On one hand, SNNs outperform ANNs in terms of latency and energy consumption [26]. Because SNNs use spike signals that consume current for a very short time, they can be implemented in hardware with very low power consumption. In addition, it is beneficial for the latency due to event-driven processing, and also more suitable than ANNs for processing sensor data that outputs spike trains, such as event-based cameras [27]. On the other hand, SNNs are generally less accurate than ANNs when processing continuous-valued data, such as MNIST [28] and CIFAR-10 [29]. One of the reasons for this is that SNNs cannot directly use backpropagation, which makes it difficult to improve their learning performance. Moreover, when SNNs handle continuous values, they need to convert continuous values into spike trains, which not only aggravates latency during inference, but also leads to lower accuracy.

To compensate for these shortcomings of SNNs, many methods have been proposed. In the following sections, we describe associated studies from two perspectives: learning strategies for SNNs (Section 3.1) and coding methods for converting continuous values to spike trains (Section 3.2).

3.1 Learning Strategy

We classify the methods to train SNNs into three categories and summarize each of them in the following sections: conversion of ANNs (Section 3.1.1), local learning rule (Section 3.1.2), and supervised learning (Section 3.1.3). Converting a trained ANN into the target SNN is the simplest and popular approach, which converts weights and activation functions in an ANN into the firing rate of an SNN. Each synaptic weights trained by local rules found in neuroscience, such as spike-timing-dependent plasticity (STDP) [30, 31] and Hebbian learning [32]. Backpropagation-based supervised learning is the most powerful tool for ANNs, however, the undifferentiable function generating spikes (Equation 2.1f) cannot be employed for SNNs merely. Many works struggle to overcome this obstacle, and some techniques achieved competitive performance of SNNs with that of ANNs.

3.1.1 Conversion of ANNs

The goal of the conversion technique is to map the same inputs and outputs of an ANN that has achieved high performance to an SNN, so that a high-performance, energy-efficient neural network can be achieved. The main advantage of the conversion approach is that it allows the use of the current widely used deep learning toolkit with very little learning overhead. The network conversion has a negligible deviation from the underlying ANN in terms of the accuracy of the SNN [19, 33], and it is also possible to provide performance guarantees that can quantify the deviation in expected accuracy [18].

Most of conversion approaches stand on the idea of rate-coding, which translates the activation of analog neurons in an ANN into the firing rate of spikes, since Pérez-Carrasco *et al.* proposed the first algorithmic way [34]. The synaptic weights are rescaled in obedience to the parameters of the spiking neurons based on leakage rate, refractory time, threshold, etc., which are set as hyperparameters.

There are several drawbacks to this conversion method. Namely not all ANNs can be converted to SNNs. First, rate-based SNNs can represent only positive values by the firing rate, while many ANNs deal with negative values as well. A principled solution to this is to use a pair of neurons, an excitatory neuron and an inhibitory neuron, for a signle neuron of ANN [34], and a practical solution for the softmax layer, common in ANNs, has also been proposed by Rueckauer *et al.* [18]. Another restriction due to the principle difference between ANNs and SNNs is that the max-pooling layer is not feasible for temporal spiking operations. Most of the researches [19, 20, 33] use mean pooling instead of max pooling to avoid this problem.

Conversion and normalization of weights may be at the expense of more spikes being generated, resulting in less energy efficient classification, and rate-coding conversions are generally not particularly efficient in terms of spikes producing. The rate-coding conversion and weight normalization may result in more spikes being generated, thus sacrificing energy efficiency. To address the inefficiency of rate-coding conversion approaches, alternative spike coding based on timing information have been utilized [35–38]. Regardless of these improvements, the conversion method cannot, in principle, perform well in spatio-temporal information processing, which is one of the potential of SNNs.

3.1.2 Local Learning Rules

Neuroscience has shown that each neuron follows local learning rules such as Hebb's rule and STDP, which have been widely used in both simulated and real environments as a neural learning mechanism for autonomous systems [39–43]. Such local learning rules generally cannot be trained well in large, deep networks, but can be implemented in small networks with high hardware-efficient learning. The STDP shows that synaptic plasticity is a phenomenon that is affected by the exact timing of each spike, especially their order [30, 31]. When a presynaptic spike precedes a postsynaptic spike, a potentiation of synaptic is strengthened, while the reverse order results in depression. In other words, nerve input that are likely to support to excitation of the neuron are strengthened, and inputs that are unlikely to contribute are weakened. Generally, STDP can be used for unsupervised learning for detecting spatio-temporal patterns as features, but to receive supervised signals, STDP can combine with global reward system e.g., reward-modulated STDP [44].

D. Hebb formulated in 1949 a fundamental principle in neuroscience that explains the adaptation of synaptic effects in the brain during the learning process, and the classical Hebb' s rule is often indicated by the shortest summary: a neuron that "fire together, wire together" [32]. The idea is expressed as

$$\Delta w_{ij} \propto v_i v_j \tag{3.1}$$

where w_{ij} is the synaptic weight between the presynaptic neuron i and the postsynaptic neuron j and v represents the activities of those neurons, respectively. The Hebbian-based learning rule, which also depends on the precise timing of pre- and post-synaptic spikes, contribute to stabilize specific neuronal activity patterns in the brain, strengthening the connectivity within assemblies of neurons that have similar characteristics [45].

With only these local learning rules, the backpropagation in deep neural networks, helping achieve high performance of the network, is difficult [46]. To flow the supervised error signals from the output layer to the input layer, a lot of studies introduce recurrent feedback connections for common feedforward architectures are not suitable for providing the training information to synapses that learn by local rules [47–51].

3.1.3 Supervised Learning With Spikes

Various approaches to solely introduce supervised learning in SNNs have been proposed. Many of these use variants of backpropagation to overcome the gap between ANNs and SNNs. The advantage of this strategy is that it can learn not only the mean-rate code like the aforementioned conversion method, but also the spatio-temporal patterns of spike trains generated by input data from event-based sensors. This approach comes at the cost of longer training times due to spiking simulation on conventional hardware, but the number of spikes required for the trained network is usually lower than for converted ones [46].

The main obstacle in backpropagation of SNNs is the undifferentiable activation function, and the approximate methods have been proposed to compute the derivatives of the loss function. SpikeProp [52] established a learning rule backpropagating temporal error signals of each neuron in the output layer, and Booij and Tat Nguyen showed its extension to multiple spike patterns [53]. However, this method is computationally expensive and has not yet been applied to recent deep learning tasks.

Although optimizing precise time spikes, the temporal learning is fragile for noise coming from various sources. An attempt has been made to avoid the influence by using the firing frequency. ReSuMe [54] are proposed to compute the derivatives of the loss function using the firing frequency, which was further extended to multilayer SNNs by Sporea and Grüning [55].

Wu *et al.* proposed a method to directly derive the spatio-temporal gradients for deep SNNs with leaky integrate and fire (LIF) neuron model [56], and Gu *et al.* provided a solution for the temporal credit assignment problem, for which SNNs need to discover the valuable features in temporally continuous data with occasional delays [24]. The authors utilized the iterative neuron model for temporal dynamics of membrane potential. To add the synapse dynamics and the filter effect into SNNs, Fang *et al.* introduced infinite impulse response (IIR) filters to add such effects and show a general algorithm for SNNs [21]. Their network remarked the state of the art for MNIST, N-MNIST [15], DVS128 gesture [16] and Australian Sign Language [57] dataset.

Overall, many supervised learning methods for SNNs have been proposed in recent years. In general, they have higher computational cost during training, but perform better than conversion and local learning rules methods. In this study, we trained our networks using the state-of-the-art supervised learning [21], which is easy to combine with ANNs that use backpropagation during training and is implemented by PyTorch [58].

3.2 Information Representation

Although the form of the spiking signal varies with the type of neurons in the real brain, SNN is based on the assumption that information is represented by spikes. We need to consider how to represent the information using spikes to input the data and readout the outputs from the network. There are two types of statistic data representation on a neuron: rate coding and temporal coding. On one hand, rate coding expresses information in terms of the frequency of firing of a neuron over a period of time, but the time interval between spikes is meaningless. On the other hand, temporal coding [59–65] represents information by the time interval of each spike.

Poisson spike sequence is usually used for the rate coding, which is simple to implement and robust to errors by using firing rate [18, 19, 66, 67]. In this case, successive spikes fire in random timing, but the overall frequency of the spikes in a certain time is predefined. However, rate coding does not take advantage of the temporal features of the spike train and generates a large number of spikes, which results in high energy consumption and long latency for inference. The time to first spike (TTFS) coding, which is one of forms of temporal coding, has been adopted in SNNs to overcome such disadvantages [60–62], where intensity of the activation is inversely proportional to the firing delay of the neuron: the one with the highest membrane potential fires first. Note that once a neuron generates a spike, it no longer generates additional spikes by applying a sufficiently long refractory period [60]. Other forms of temporal coding use the inter-spike interval (ISI), the precise delay between consecutive spikes, to convert the intensity of the activation [63–65]. Temporal coding uses fewer spikes and improve overall performance in terms of latency and efficiency, however, it requires more complex implementation. In addition, for image classification, each neurons in the output layer simply integrate its receiving spikes during a certain time to decide the inferred labels.

Although how information is represented using spikes in the biological brain is still discussed [64, 68], we focused on rate coding since such coding is more commonly used than temporal coding for converting continuous-valued input to event-driven data before the first layer on an SNN.

Chapter 4

Proposed Framework

In this chapter, we depicted the proposed hybrid neural networks (HNNs), having ALs, SLs and a coding module, from the aspects of the learning and coding. There are two learning approaches for SNNs: learning ALs and SLs either separately or simultaneously, the details of which are described in Section 4.1. Because the format of data is depending on the types of the neural layers, the data should be converted from continuous values into spike trains at the boundary. In Section 4.2, three types of coding methods were explained.

4.1 Optimization of HNNs

There are two learning methods we proposed for the HNNs: to learn ALs and SLs of an HNN separately and to learn simultaneously. On one hand, an ANN was trained in advance, and when building an HNN, the corresponding ALs were taken out and connected to the following SLs while keeping the ALs' weights. As Section 4.1.1 showing, when learning the HNN, the weights of the extracted ALs are fixed, and only the weights of the SLs are updated. In this type of network, since ALs do not need to learn with the SLs, non-differentiable coding functions can also be used to convert the latent vectors from the last AL into spikes, and also this kind of network takes less time to learn than the latter.

On the other hand, both ALs and SLs are combined as an HNN from the beginning, and all the layers are learned from scratch by backpropagation (Section 4.1.2). At this point, between the last AL and the first SL, coding needs to take place, and it should be differentiable for backpropagation. Therefore, undifferentiable coding methods are not available in spatio-temporal backpropagation.

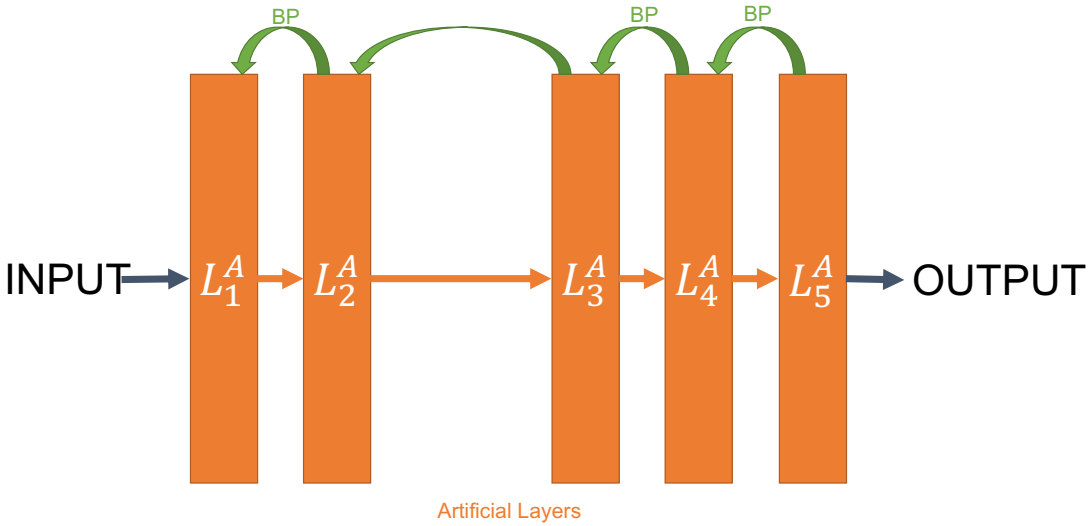
4.1.1 Separate Learning

In this learning method, ALs and SLs are not learned simultaneously, but in two stages. Initially, as the first stage of learning, we trained a pure ANN with ALs in all layers (Figure 4.1(a)). The structure of this ANN and the number of neurons are the same as the HNN that is created in the next step. The first stage of training is to learn the ANN using the training dataset for a set number of epochs, and to evaluate the network at each epoch using the validation dataset.

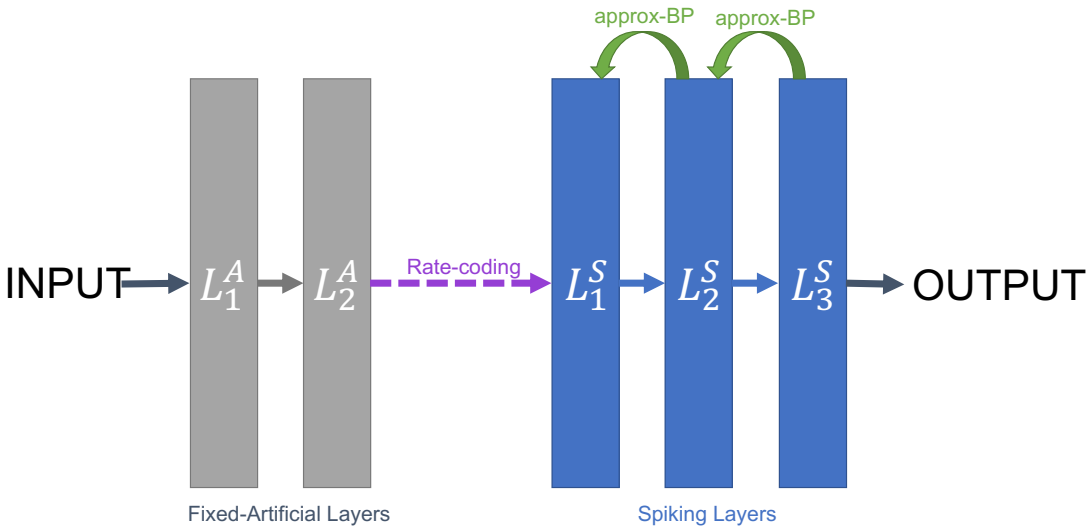
The next step is to construct an HNN with ALs and SLs on the same architecture as the trained ANN. At this time, the weights of ALs are set to the weights that are the

same one on the highest-accuracy ANN in the first phase. In the second stage of learning (Figure 4.1(b)), the weights of ALs are fixed and only SLs are trained. The configuration of the dataset used in this case is the same as the one used in the first stage. In the second stage, as in the first stage, we trained on the training dataset and use the validation dataset to determine the weights of the HNN to be used during testing. Between the last AL and the first SL (L_2^A and L_1^S in Figure 4.1(b) respectively), the coding process described in Section 4.2 is performed to convert the continuous values into spike signals. The accuracy using the test dataset in this second training was used to compare the performance of this HNN with other networks in Chapter 5.

By learning an ANN once, when a developer wants to change the ratio of ALs to SLs in an HNN, he only need to learn SLs on the second phase, which is advantageous in actual operation. It is also possible to use the pretrained ANNs which are currently published from many organizations.



(a) The first stage of separate learning



(b) The second stage of separate learning

Figure 4.1: Separate learning.

4.1.2 Simultaneous Learning

Unlike the separate learning in the previous section, this learning method trains ALs and SLs on an HNN together using usual and approximated backpropagation [21]. In the simultaneous learning (Figure 4.2), information represented by continuous values or spike trains is also passed between ALs and SLs by adding a coding layer on the interface. However, since this learning method applies the chain rule through ALs and SLs, the coding must also be differentiable, and one of coding methods we used are not applicable e.g., Poisson coding in Section 4.2.3.

This method can be expected to handle more complex tasks than the separate learning because ALs and SLs can collaborate to solve the tasks. In addition, the overall training time is shorter when training a single HNN, since it does not need to train an whole ANN as in the separate learning.

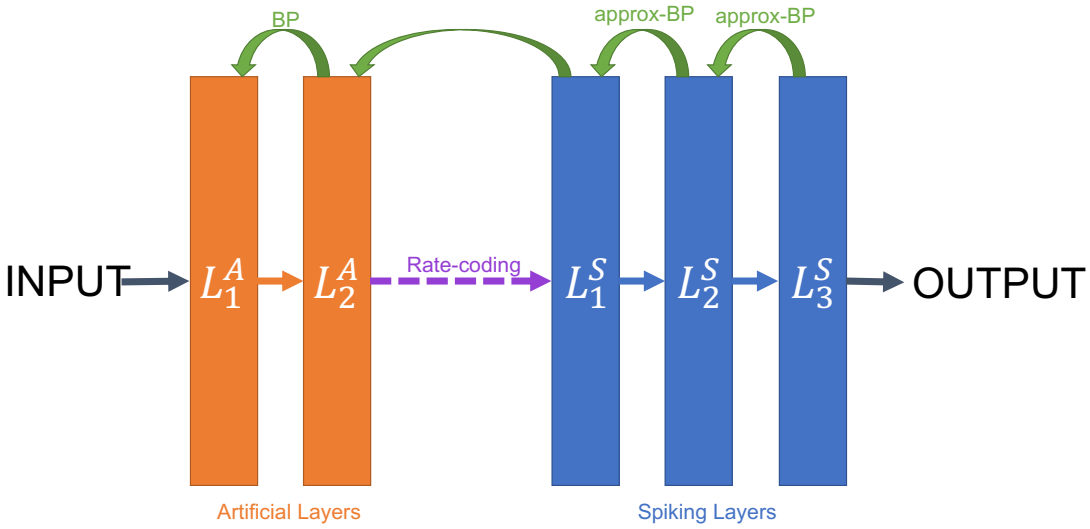


Figure 4.2: Simultaneous learning.

4.2 Rate-based Coding

Although neuromorphic sensors generate spiking data which are appropriate for SNNs, many conventional datasets need to be converted to spiking data. The proposed architecture of the HNNs has to process data as continuous values and spike trains. To convert continuous values into spike trains, rate-based coding is utilized in the interface between artificial and spiking layers.

The performances of all the networks in this thesis are evaluated with image input consisted of continuous values, so ALs calculate the continuous-valued output from the input directly but the spiking layers require additional time dimension which the original input data do not have. To solve this problem, we explored three methods: duplicate (Section 4.2.1), Gaussian (Section 4.2.2) and Poisson (Section 4.2.3) coding.

4.2.1 Duplicate Coding

This is the simplest way for rate-based coding: expanding the output vector $\boldsymbol{a}$ along the time axis. This continuous value trains are the input of spiking neurons in the first SL instead of a spiking train. The input for the first SL, which refers to the coded output of the last AL O_i^{AL}, is:

$$O_i^{AL}[t] = a_i \tag{4.1}$$

4.2.2 Gaussian Coding

The duplicating method codes a continuous value to a continuous-valued train deterministically, while biological networks are stochastic due to uncertainty factors in nature. This method stochastically converts the continuous values generated from artificial layer output with reparameterization trick [69] for connectivity in backpropagation.

The last AL outputs $\boldsymbol{\mu}$ and $\ln \boldsymbol{\sigma}$ instead of a vector, and the first SL receives reparameterized continuous-valued trains $O_i^{AL}[t] \sim \mathcal{N}(\mu, \sigma^2)$ (Figure 4.5). By reparameterization trick, the input of the first SL is defined as:

$$O_i^{AL}[t] = \mu_i + \sigma_i \varepsilon \tag{4.2}$$

where ε is an auxiliary noise variable $\varepsilon \sim \mathcal{N}(0, 1)$.

The reparameterization trick can make the boundary between AL and SL differentiable, however, requiring twice the number of neurons in the last AL compared with other coding methods in order to measure both of the performance fairly. Thus, although this coding method can be used in any forms of networks theoretically, SLs empirically do not be connected with fixed ALs in this way.

4.2.3 Poisson Coding

Poisson coding, which is based on a conception in probability theory known as the Poisson process, is a widely used coding method [41, 70, 71], which generates a spike train from a continuous value according to Poisson distribution, which only expand a continuous value along time dimension stochastically.

With Poisson coding, the converting function outputs a sequence of spikes so that the time difference between spikes follows a Poisson distribution, which is not differentiable unlike the above two methods (Sections 4.2.1 and 4.2.2). Thus, the Poisson coding cannot be used with the simultaneous learning method.

4.3 Proposed Networks

As described in the previous sections, the two learning policies, separate and simultaneous, and the three coding methods, duplicating, Gaussian and Poisson coding, have limited combinations due to their traits, and in this study, we investigated the possibility of HNNs with four methods. For simplicity, we assume that all networks have a feedforward architecture.

The main limiting factor to develop an HNN is whether the coding method is differentiable or not. If fixed ALs are utilized in the HNN, all coding methods can be adopted. On the other hand, if trainable ALs are used, only differentiable methods can be employed in

HNNs. Moreover, from the evaluation viewpoint, the Gaussian coding implemented by the reparameterization trick was not combined with HNNs having fixed ALs, since the reparameterization trick implementation requires more neurons than the basic network architecture.

The trainable-AL-and-SL-with-duplicate-coding network, shown in Figure 4.3, has ALs and SLs, both of which are trained simultaneously through backpropagation, using the duplicating method as the coding after the last AL. Fixed-AL-and-trainable-SL-with-duplicating-coding network (Figure 4.4) optimizes the weights only on SLs during training, and for adding time dimension between AL and SL, the network simply copies the output from the last AL for a certain number of time steps. As shown in Figure 4.5, the trainable-AL-and-SL-with-Gaussian-coding network is trained for the whole layers together while training phase even with a stochastic coding following Gaussian distribution. And this network takes the longest time to train in the four types in our thesis because unlike networks with fixed ALs, the whole layers should be trained and the reparameterization trick used in Gaussian coding is more computationally complex than simply duplicating. In fixed-AL-and-trainable-SL-with-Poisson-coding network (Figure 4.6), Poisson coding are used for coding to generate spike trains from the latent vector rendered from fixed ALs. Due to the undifferentiable coding method, only SLs are trained.

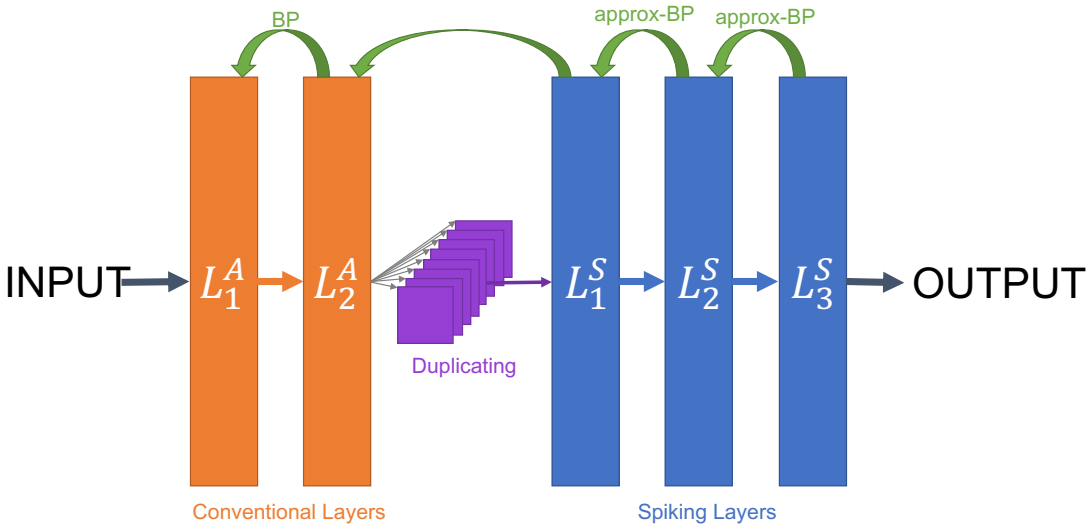


Figure 4.3: Trainable-AL-and-SL with duplicate coding

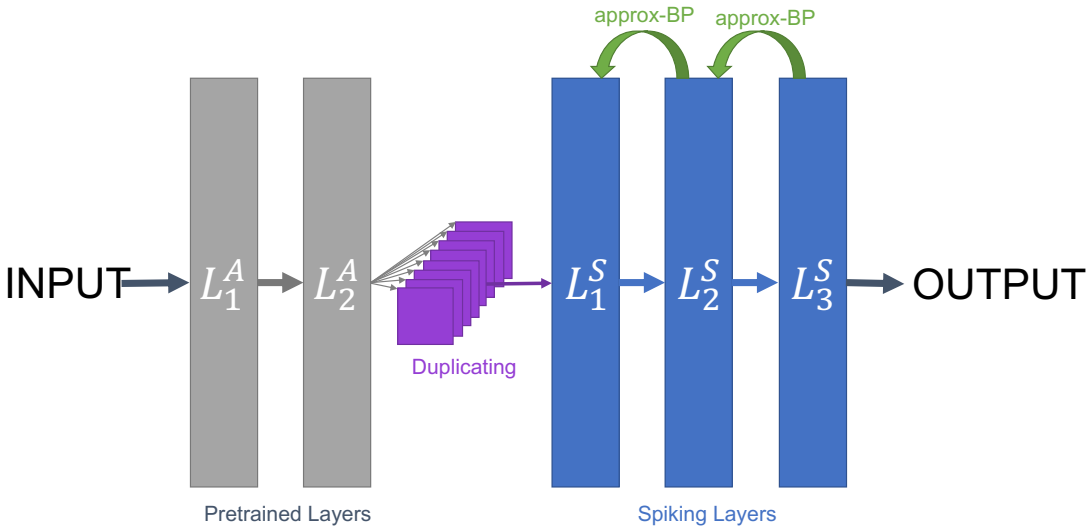


Figure 4.4: Fixed-AL-and-trainable-SL with duplicate coding

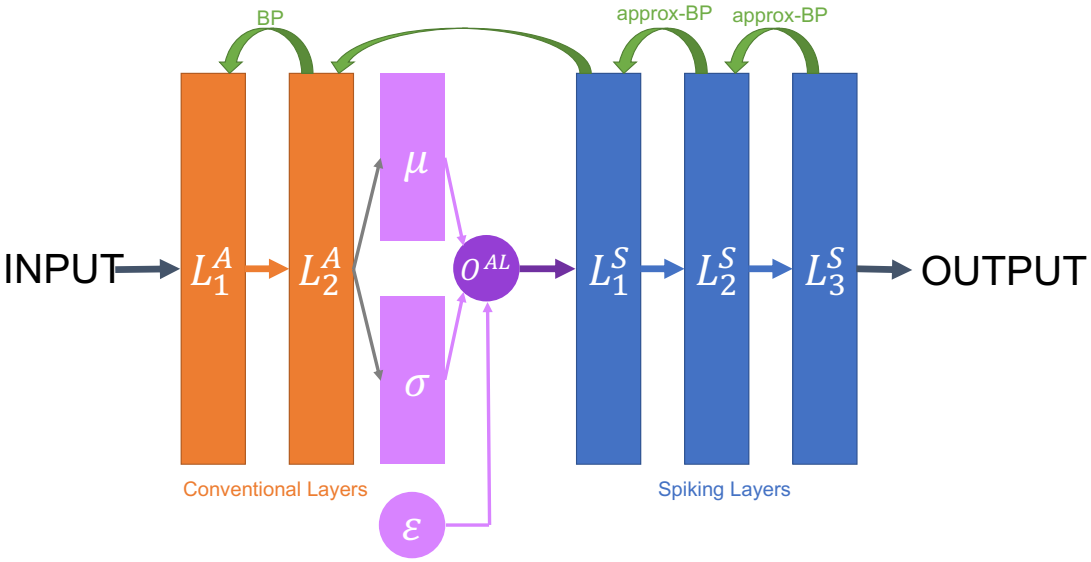


Figure 4.5: Trainable-AL-and-SL with Gaussian coding

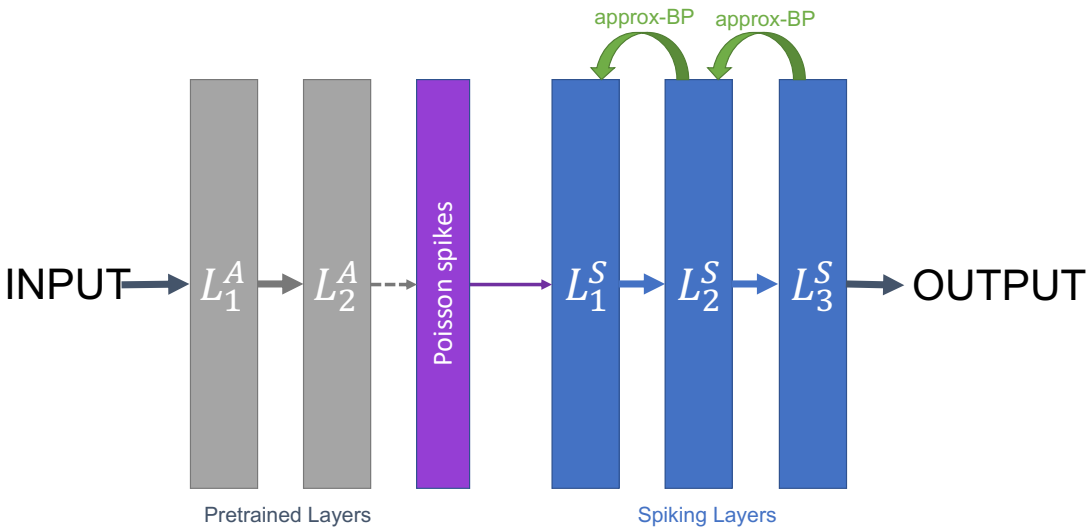


Figure 4.6: Fixed-AL-and-trainable-SL with Poisson coding

Chapter 5

Experiments

In this chapter, the aforementioned HNNs are evaluated with various architectures for image classification with the setting shown in Section 5.1. Initially, we examined the minor differences in the methods of inputting coded data into SL in Section 5.2. The results were validated in terms of coding and learning algorithms for multilayer perceptron (MLP) and convolutional neural network (CNN) architectures while varying the ratio of AL to SL and show the general tendencies of HNNs (Section 5.3).

5.1 Experimental Settings

In every experiment on this thesis, two datasets, MNIST and CIFAR-10, were used for measuring the performance of each network. Each dataset was divided into three subsets; namely the training, validation and testing data with a ratio of $6:2:2$.

All the HNNs with the parameters shown in 5.1 are trained using Adam optimizer [72] with multi-step decay, decreasing the learning rate by a factor of 10 at 50 % and 75 % of the number of epochs. Each network was trained with the training dataset for 100 and 150 epochs on MNIST and CIFAR-10, respectively. At the end of each epoch, the performance of a network is evaluated with the validation dataset, and the network achieving the best performance on the validation dataset was finally used to compare the performance based on the test dataset.

Table 5.1: Network parameters

	MLP	CNN
batch size	64	32
epoch	100	150
initial learning rate	0.0001	0.0001
time length T	25	25
τ_m	4	4
τ_s	1	1
V_{th}	1	1
θ	$\mathrm{e}^{-\frac{1}{4}}$	$\mathrm{e}^{-\frac{1}{4}}$
dataset	MNIST [28]	CIFAR-10 [29]

5.2 Receiving Continuous Values

There are two ways how a continuous value train is inserted into spiking neurons: through the axon or not. The former is used in the first study of IIR-based SNNs [21], in which the approximate backpropagation method we utilized was proposed. On one hand, the continuous-valued input is converted to postsynaptic potential (PSP) through the axon, being treated in the same way as spike trains, and spiking neurons in the first SL receive the PSP. On the other hand, in the second way spiking neurons in the first SL directly receive the continuous-valued input instead of PSP. This is more bio-plausible because the sensory nerve endings in animals are sensory neurons, rather than axons.

In the experiment for MNIST with MLP(`S784-S500-S10`), the axon-input network achieved 97.56 %, and the direct-input network 97.71 %. The results reveal that the axon-inputting method cannot contribute to the improvement of the performance of networks. Therefore, we used the direct-input method in Section 5.3.

5.3 Hybrid Neural Networks

In this section, we tested the HNNs we proposed on image classification tasks: MNIST and CIFAR-10. Similarly, the HNNs can be regulated with two aspects: rate-based coding, including duplicating, Gaussian and Poisson coding method, and neural layers, where SLs are combined with fixed ALs or trainable ALs. We note that the fixed-AL-and-trainable-SL network with Gaussian coding is not fairly compared with others, because it relies on reparameterization trick and requires additional neurons. In the fixed-AL-and-trainable-SL networks, the fixed ALs are generated from the pure ANN, which was the model with the highest accuracy when trained on the training dataset and evaluated on the validation dataset.

In Tables 5.2–5.7, the network architectures are shown by a notation, in which `A`n and `S`m indicate a layer having n artificial neurons and m spiking neurons, respectively. Additionally, `A`n`C`k or `S`n`C`k indicate a artificial/spiking convolutional layer with the kernel $k \times k$. Here a coding method is implicitly inserted between `A`n`-S`m.

To evaluate HNNs, we constructed two types of architectures: three-layer multilayer perceptron (MLP) and convolutional neural network (CNN) for MNIST and CIFAR-10, respectively As the baselines, pure SNNs used dual exponential postsynaptic potential kernel with the same architectures were trained. Moreover, the last rows in Tables 5.2–5.7 are the results of the pure ANN that is fully consisted of ALs. Here, the respective results of pure SNNs and ANNs with separate learning are not described in all of the tables because they do not need to train separately each layer.

5.3.1 Separate versus Simultaneous Training

To compare separate and simultaneous learning for classification accuracy, we conducted comparison experiments using the above networks with duplicate coding, which can be used for both types of learning. Tables 5.2 and 5.3 shows the results of MLPs and CNNs, each type of networks having the same number of neurons but the ratio of ALs to SLs is different, respectively. In each table, the accuracy for the pure SNN and ANN with fixed ALs are not listed, because both types of pure networks do not have any layer for pretraining. Note

that the results of the pure SNN and ANN are similar with the ones in Tables 5.4 and 5.5 since the networks having the same architectures and using duplicate coding.

Those tables also elucidate that using pure networks achieves higher accuracy. Also, in the HNNs, increasing the percentage of ALs improves the performance, which shows the trade-off between computational accuracy and energy efficiency; in rate-based coding, an artificial neuron can represent the information more accuracy but generally a spiking neuron can reduce power consumption. Furthermore, while the networks with fixed ALs show higher accuracy than ones with trainable ALs, the performance of the more complex tasks (CIFAR-10) was conversely reduced when the ratio of SLs in a network to be trained is too small. Thus, in complex tasks with huge networks, learning ALs and SLs at the same time give better accuracy than not only using fixed ALs but the pure SNN.

Table 5.2: MLPs with duplicate coding for MNIST

	Traning method	
	Separate	Simultaneous
`S784-S500-S10`	-	98.13 %
`A784-S500-S10`	98.22 %	97.78 %
`A784-A500-S10`	99.17 %	98.07 %
`A784-A500-A10`	-	98.23 %

Table 5.3: CNNs with duplicate coding for CIFAR-10

	Traning method	
	Separate	Simultaneous
`S32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10`	-	59.96 %
`A32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10`	56.01 %	53.23 %
`A32C3-A32C3-A64C3-P2-S64C3-P2-S512-S10`	63.30 %	55.22 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-S512-S10`	62.78 %	66.08 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-A512-A10`	-	70.09 %

5.3.2 Gaussian Coding

The results of MLPs for MNIST and CNNs for CIFAR-10 with Gaussian coding are shown in Tables 5.4 and 5.5. As mentioned in the above section, Gaussian coding can be employed with only trainable ALs due to the additional neurons for the reparameterization trick.

Here `S784-S500-S10` in Table 5.4 and `S32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10` in Table 5.5 coded the input data into spiking trains with duplicating method since it is not possible to use Gaussian coding instead of duplicate coding without varying the network architecture, and the accuracies of both networks shown in each table as the baseline are similar with Tables 5.2 and 5.3 respectively. Moreover, `A784-A500-A10` in Table 5.4 and `A32C3-A32C3-A64C3-P2-A64C3-P2-A512-A10` in Table 5.5 listed for only comparison are not have the coding layer; these networks directly infer from original data, and the results on both tables also same with Tables 5.2 and 5.3 respectively.

From both of the results (Figures 5.3 and 5.4), with the exception of `A32C3-A32C3-A64C3-P2-S64C3-P2-S512-S10` network, the trend of improving accuracy with increasing the ration of AL is maintained from duplicate coding (Section 5.3.1). In addition, comparing

with trainable-AL-and-SL model with duplicate coding on Tables 5.2 and 5.3 accuracies of the models with Gaussian coding on Tables 5.3 and 5.4 are higher in lower ratio of ALs.

Table 5.4: MLPs with Gaussian coding for MNIST

Network architecture	Accuracy
`S784-S500-S10` (baseline)	98.13 %
`A784-S500-S10`	98.02 %
`A784-A500-S10`	98.03 %
`A784-A500-A10`	98.23 %

Table 5.5: CNNs with Gaussian coding for CIFAR-10

Network architecture	Accuracy
`S32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10` (baseline)	59.96 %
`A32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10`	59.35 %
`A32C3-A32C3-A64C3-P2-S64C3-P2-S512-S10`	65.30 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-S512-S10`	63.82 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-A512-A10`	70.09 %

5.3.3 Poisson Coding

Poisson coding is the way to generate spike trains following Poisson distribution from continuous values, which is known as a mathematical technic, the Poisson process. The Poisson process is usually not differentiable, so the trainable-AL-and-SL cannot be used with that coding. Additionally, to apply Poisson coding, the continuous values should be normalized into $[0, 1]$. In this experiment, for simplifying, we used the Sigmoid function, the output values are $[0, 1]$, as the activation function instead of ReLU, which are utilized in the above, in the pretrained model and ALs.

As shown in Tables 5.6 and 5.7, using Poisson coding instead of duplicate or Gaussian coding dramatically decreased the performance of every network architectures. Unlike previous results, the continuous-valued data were converted into spike trains with Poisson coding for pure SNNs, `S784-S500-S10` on Table 5.6 and `S32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10` on Table 5.7. The last row of each table show the result of pretrained ANN, which is the basis of ALs in HNNs.

The results on Table 5.6 show that even such low accuracies generally follow the tendency, more ALs lead to higher accuracy. On the other hand, the results on Table5.7 shows the opposite tendency, and it would be said that Poisson coding generates too much noise, which leads to decreases the accuracy rather than improving the generalization performance. That noisy latent vectors are seen to reduce the accuracy of image recognition.

Table 5.6: MLPs with Poisson coding for MNIST

Network architecture	Accuracy
`S784-S500-S10` (baseline)	98.09 %
`A784-S500-S10`	89.67 %
`A784-A500-S10`	95.93 %
`A784-A500-A10` (pretrained)	97.67 %

Table 5.7: CNNs with Poisson coding for CIFAR-10

Network architecture	Accuracy
`S32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10` (baseline)	59.83 %
`A32C3-S32C3-S64C3-P2-S64C3-P2-S512-S10`	50.28 %
`A32C3-A32C3-A64C3-P2-S64C3-P2-S512-S10`	36.94 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-S512-S10`	31.32 %
`A32C3-A32C3-A64C3-P2-A64C3-P2-A512-A10` (pretrained)	56.43 %

5.4 Summary

Empirical results (Figures 5.2–5.7) show a general tendency for accuracy to improve as the percentage of AL in a hybrid model increased, with the exception of two-AL-and-three-SL CNNs (`A32C3-A32C3-A64C3-P2-S64C3-P2-S512-S10`). This confirms our hypothesis that ANNs are better suited to handle continuous values.

The highest scores of the HNNs with the Gaussian coding are not greater than that with the duplicate coding, however, networks with fewer ALs in Tables 5.4 and 5.5 performed better than trainable-AL-and-SL networks with the duplicate coding, especially for CIFAR-10. From this, the Gaussian coding shows an advantage in the following experiments that high performance can be expected in more complex tasks even with fewer ALs.

In these experiments, we did not find any advantage in the widely used Poisson coding. In particular, the coding method being used in an HNN, its accuracy is greatly reduced. This is probably because there is too much information missing due to noise generated by such coding process. On the other hand, pure SNNs show competitive accuracy as other coding methods, as shown in previous studies [26]. Interestingly, about HNNs having fixed ALs in Figure 5.7, the CNNs with fewer ALs outperform CNNs with more ALs. This suggests that the use of a spiking convolutional layer immediately after the coding process contributes to the performance improvement of the network because this inclination is not seen in the MLP results.

Chapter 6

Conclusions and Future Work

In this study, we introduced the concept of the HNN that combines an ANN, the currently widely used floating-point neural network, with an SNN, more bio-plausible, event-driven neural network and evaluated the performance of different network configurations of it using image classification tasks. The results show that increasing the percentage of ALs on an HNN tends to improve the classification accuracy, and that simultaneous learning is more advantageous for more complex tasks. Several HNNs exceeded the performance of pure SNNs as the baseline and showed their effectiveness. Moreover, it was clarified that the performance changes depending on the coding methods used to transform continuous values to spiking trains for connecting an AL and a SL and that even if the simplest duplicate coding is chosen, the performance is equal to or better than other coding methods on MNIST. However, on the more complex dataset, the Gaussian coding contributes the accuracy more even with lower ratios of ALs in an HNN.

In general, SLs are more energy efficient than ALs for the hardware implementation because they use spikes which flow current in very short time [26]. In this thesis, we focus on the conceptual verification of HNN and exploring the improvement of performance over SNNs, and we do not evaluate the energy consumption because there is no hardware implementation device of HNN and we do not discuss the estimated energy consumption of SNN [73, 74]. Certainly, the hardware-implemented SNNs can dramatically improve the energy efficient for training and inferring, but the energy efficiency is highly depending on the kinds of chips [75–79]. Therefore, as a future work, it is worthwhile to quantitatively investigate the trade-off between classification accuracy and energy consumption when the number of ALs is increased.

In addition, *separately and simultaneous learning* can be considered as the third learning method for HNN that combines separate learning and simultaneous learning. In this method, ALs learned in the first stage are combined with SLs in the second stage, like separate learning, and then further learning is performed. Although the third learning method requires a longer overall training time, it is expected to provide higher performance for a wider range of tasks, since fixed-AL-and-trainable-SL networks may have high accuracy.

To invest more details of the HNN, comprehensive experiments with different optimization algorithms and larger datasets are needed. For deep neural networks, a large number of optimization algorithms are proposed [80–98], which are also employed for SNNs instead of Adam [72]. Like other works [70, 99–101], this thesis also used MNIST and CIFAR-10 to validate the performance of networks, but the tendency was different depending on the

dataset, so it needs to be validated with more various data distributions, such as Fashion-MNIST [102], EMNIST [103], CIFAR-100 and ImageNet [104]. It is necessary to analyze the characteristics of HNNs to further improve their performance for the future development of SNNs for utilizing the current ANN technology and knowledge. We believe that our work can contribute to further increasing the potential of SNNs, which are more energy-efficient than ANNs but disadvantageous for handling continuous-valued data output.

Acknowledgement

First and foremost, I would like to express my sincere gratitude and appreciation to Dr. Satoh Tetsuji and Dr. Hai-Tao Yu for offering the opportunity to pursue my master under their supervision. Dr. Satoh Tetsuji has been an incredible academic guider that enlightened and encouraged my research topics constantly. I would like to thank Dr. Hai-Tao Yu for being my mentor and providing me valuable suggestions and solid support about my research. Finally, my dearest thanks go to my wife, Mai (right-side in Figure 6.1), who has stood by me through all my travails, my absences, my fits of pique and impatience. She has been always supporting me and improving my potential with her wordless love, care and understanding.

Figure 6.1: My wife and me.

References

[1] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional Neural Network Architectures for Matching Natural Language Sentences. *arXiv:1503.03244 [cs]*, March 2015.

[2] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent Trends in Deep Learning Based Natural Language Processing. *arXiv:1708.02709 [cs]*, November 2018.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020.

[4] Haotian Liu, Rafael A. Rivera Soto, Fanyi Xiao, and Yong Jae Lee. YolactEdge: Real-time Instance Segmentation on the Edge (Jetson AGX Xavier: 30 FPS, RTX 2080 Ti: 170 FPS). *arXiv:2012.12259 [cs]*, December 2020.

[5] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-Aware Minimization for Efficiently Improving Generalization. *arXiv:2010.01412 [cs, stat]*, December 2020.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*, February 2015.

[8] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D Mesh Renderer. *arXiv:1711.07566 [cs]*, November 2017.

[9] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel,

and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, Vol. 529, No. 7587, pp. 484–489, January 2016.

[10] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An Analysis of Deep Neural Network Models for Practical Applications. *arXiv:1605.07678 [cs]*, April 2017.

[11] Jizhizi Li, Jing Zhang, Stephen J. Maybank, and Dacheng Tao. End-to-end Animal Image Matting. *arXiv:2010.16188 [cs, eess]*, October 2020.

[12] Daniel Drubach. *The Brain Explained*. Prentice Hall, 2000.

[13] Ali Samadzadeh, Fatemeh Sadat Tabatabaei Far, Ali Javadi, Ahmad Nickabadi, and Morteza Haghir Chehreghani. Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction. *arXiv:2003.12346 [cs]*, March 2020.

[14] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, Vol. 10, No. 9, pp. 1659–1671, 1997.

[15] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, Vol. 9, p. 437, 2015.

[16] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] Qiang Yu, Chenxiang Ma, Shiming Song, Gaoyan Zhang, Jianwu Dang, and Kay Chen Tan. Constructing Accurate and Efficient Deep Spiking Neural Networks with Double-threshold and Augmented Schemes. *arXiv:2005.03231 [cs]*, May 2020.

[18] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, Vol. 11, p. 682, 2017.

[19] Peter U. Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2015-Septe. Institute of Electrical and Electronics Engineers Inc., September 2015.

[20] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *International Journal of Computer Vision*, Vol. 113, No. 1, pp. 54–66, May 2015.

[21] Haowen Fang, Amar Shrestha, Ziyi Zhao, and Qinru Qiu. Exploiting Neuron and Synapse Filter Dynamics in Spatial Temporal Learning of Deep Spiking Neural Network. *arXiv:2003.02944 [cs, stat]*, July 2020.

[22] R. B. Stein. A THEORETICAL ANALYSIS OF NEURONAL VARIABILITY. *Biophysical Journal*, Vol. 5, pp. 173–194, March 1965.

[23] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv:1705.06963 [cs]*, May 2017.

[24] Pengjie Gu, Rong Xiao, Gang Pan, and Huajin Tang. STCA: Spatio-temporal credit assignment with delayed feedback in deep spiking neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 1366–1372. International Joint Conferences on Artificial Intelligence Organization, July 2019.

[25] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate Gradient Learning in Spiking Neural Networks. *arXiv:1901.09948 [cs, q-bio]*, May 2019.

[26] Lei Deng, Yujie Wu, Xing Hu, Ling Liang, Yufei Ding, Guoqi Li, Guangshe Zhao, Peng Li, and Yuan Xie. Rethinking the performance comparison between SNNS and ANNS. *Neural Networks*, Vol. 121, pp. 294–307, January 2020.

[27] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Joerg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based Vision: A Survey. *arXiv:1904.08405 [cs]*, August 2020.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, 1998.

[29] Alex Krizhevsky and Geoffrey Hinton. *Learning Multiple Layers of Features from Tiny Images*. PhD thesis, Citeseer, 2009.

[30] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of Synaptic Efficacy by Coincidence of Postsynaptic APs and EPSPs. *Science*, Vol. 275, No. 5297, pp. 213–215, January 1997.

[31] Guo-qiang Bi and Mu-ming Poo. Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type. *Journal of Neuroscience*, Vol. 18, No. 24, pp. 10464–10472, December 1998.

[32] Donald Olding Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, April 2005.

[33] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going Deeper in Spiking Neural Networks: VGG and Residual Architectures. *arXiv:1802.02627 [cs]*, February 2019.

[34] J. A. Pérez-Carrasco, B. Zhao, C. Serrano, B. Acha, T. Serrano-Gotarredona, S. Chen, and B. Linares-Barranco. Mapping from Frame-Driven to Frame-Free Event-Driven Vision Systems by Low-Rate Rate Coding and Coincidence Processing–Application to Feedforward ConvNets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 11, pp. 2706–2719, November 2013.

[35] Garrick Orchard, Cedric Meyer, Ralph Etienne-Cummings, Christoph Posch, Nitish Thakor, and Ryad Benosman. HFirst: A Temporal Approach to Object Recognition. *arXiv:1508.01176 [cs]*, August 2015.

[36] X. Lagorce, G. Orchard, F. Galluppi, B. E. Shi, and R. B. Benosman. HOTS: A Hierarchy of Event-Based Time-Surfaces for Pattern Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, No. 7, pp. 1346–1359, July 2017.

[37] Amos Sironi, Manuele Brambilla, Nicolas Bourdis, Xavier Lagorce, and Ryad Benosman. HATS: Histograms of averaged time surfaces for robust event-based object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[38] Davide Zambrano and Sander M. Bohte. Fast and Efficient Asynchronous Neural Computation with Adapting Spiking Neural Networks. *arXiv:1609.02053 [cs]*, September 2016.

[39] J. Nielsen and H. H. Lund. Spiking neural building block robot with Hebbian learning. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, Vol. 2, pp. 1363–1369 vol.2, 2003.

[40] T. S. Clawson, S. Ferrari, S. B. Fuller, and R. J. Wood. Spiking neural network (SNN) control of a flapping insect-scale robot. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 3381–3388, 2016.

[41] Zhenshan Bing. *Biological-Inspired Hierarchical Control of a Snake-like Robot for Autonomous Locomotion*. Dissertation, Technische Universität München, München, 2019.

[42] Sebastian Glatz, Julien Martel, Raphaela Kreiser, Ning Qiao, and Yulia Sandamirskaya. Adaptive motor control and learning in a spiking neural network realised on a mixed-signal neuromorphic processor. In *Proceedings - IEEE International Conference on Robotics and Automation*, Vol. 2019-May, pp. 9631–9637. Institute of Electrical and Electronics Engineers Inc., May 2019.

[43] Zhenshan Bing, Claus Meschede, Kai Huang, Guang Chen, Florian Rohrbein, Mahmoud Akl, and Alois Knoll. End to End Learning of Spiking Neural Network Based on R-STDP for a Lane Keeping Vehicle. In *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4725–4732. Institute of Electrical and Electronics Engineers Inc., September 2018.

[44] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in Neural Circuits*, Vol. 9, p. 85, 2016.

[45] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014.

[46] Michael Pfeiffer and Thomas Pfeil. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience*, Vol. 12, p. 774, October 2018.

[47] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in Neuroscience*, Vol. 12, p. 608, 2018.

[48] Nikola T. Markov, Julien Vezoli, Pascal Chameau, Arnaud Falchier, René Quilodran, Cyril Huissoud, Camille Lamy, Pierre Misery, Pascale Giroud, Shimon Ullman, Pascal Barone, Colette Dehay, Kenneth Knoblauch, and Henry Kennedy. Anatomy of hierarchy: Feedforward and feedback pathways in macaque visual cortex. *Journal of Comparative Neurology*, Vol. 522, No. 1, pp. 225–259, 2014.

[49] Emre O. Neftci, Charles Augustine, Somnath Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, Vol. 11, p. 324, 2017.

[50] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. Towards deep learning with segregated dendrites. *eLife*, Vol. 6, p. e22901, December 2017.

[51] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards Biologically Plausible Deep Learning. February 2015.

[52] Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, Vol. 48, No. 1-4, pp. 17–37, October 2002.

[53] Olaf Booij and Hieu tat Nguyen. A gradient descent rule for spiking neurons emitting multiple spikes. *Information Processing Letters*, Vol. 95, No. 6, pp. 552–558, September 2005.

[54] Filip Ponulak and Andrzej Kasiński. Supervised Learning in Spiking Neural Networks with ReSuMe: Sequence Learning, Classification, and Spike Shifting. *Neural Computation*, Vol. 22, No. 2, pp. 467–510, October 2009.

[55] Ioana Sporea and André Grüning. Supervised Learning in Multilayer Spiking Neural Networks. *Neural Computation*, Vol. 25, No. 2, pp. 473–509, February 2013.

[56] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, Vol. 12, p. 331, 2018.

[57] Mohammed Waleed Kadous, et al. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. University of New South Wales Kensington, October 2002.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

[59] Iulia M. Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function: Learning with Backpropagation. *arXiv:1907.13223 [cs, q-bio]*, November 2020.

[60] Lei Zhang, Shengyuan Zhou, Tian Zhi, Zidong Du, and Yunji Chen. TDSNN: From Deep Neural Networks to Deep Spike Neural Networks with Temporal-Coding. *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, No. 01, pp. 1319–1326, July 2019.

[61] B. Rueckauer and S. Liu. Conversion of analog to spiking neural networks using sparse temporal coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, May 2018.

[62] Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2FSNN: Deep Spiking Neural Networks with Time-to-first-spike Coding. *arXiv:2003.11741 [cs, stat]*, March 2020.

[63] C. Chien, S. Liu, and A. Steimer. A Neuromorphic VLSI Circuit for Spike-Based Random Sampling. *IEEE Transactions on Emerging Topics in Computing*, Vol. 6, No. 1, pp. 135–144, January 2018.

[64] Arvind Kumar, Stefan Rotter, and Ad Aertsen. Spiking activity propagation in neuronal networks: Reconciling different perspectives on neural coding. *Nature Reviews Neuroscience*, Vol. 11, No. 9, pp. 615–627, September 2010.

[65] T. Mesquida, A. Valentian, D. Bol, and E. Beigne. Impact of the AER-induced timing distortion on Spiking Neural Networks implementing DSP. In *2016 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, pp. 1–4, June 2016.

[66] Seijoon Kim, Seongsik Park, Byunggook Na, and Sungroh Yoon. Spiking-YOLO: Spiking Neural Network for Energy-Efficient Object Detection. *arXiv:1903.06530 [cs, stat]*, November 2019.

[67] E. D. Adrian. The impulses produced by sensory nerve endings: Part I. *The Journal of Physiology*, Vol. 61, No. 1, pp. 49–72, March 1926.

[68] Rufin Van Rullen and Simon J. Thorpe. Rate Coding Versus Temporal Order Coding: What the Retinal Ganglion Cells Tell the Visual Cortex. *Neural Computation*, Vol. 13, No. 6, pp. 1255–1283, June 2001.

[69] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014.

[70] Peter Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, Vol. 9, p. 99, 2015.

[71] Damien Querlioz, Olivier Bichler, Philippe Dollfus, and Christian Gamrat. Immunity to Device Variations in a Spiking Neural Network With Memristive Nanodevices. *IEEE Transactions on Nanotechnology*, Vol. 12, No. 3, pp. 288–295, May 2013.

[72] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017.

[73] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition. *International Journal of Computer Vision*, Vol. 113, No. 1, pp. 54–66, May 2015.

[74] Nitin Rathi and Kaushik Roy. DIET-SNN: Direct Input Encoding With Leakage and Threshold Optimization in Deep Spiking Neural Networks. *arXiv:2008.03658 [cs, stat]*, December 2020.

[75] Dongseok Kwon, Suhwan Lim, Jong-Ho Bae, Sung-Tae Lee, Hyeongsu Kim, Young-Tak Seo, Seongbin Oh, Jangsaeng Kim, Kyuho Yeom, Byung-Gook Park, and Jong-Ho Lee. On-chip training spiking neural networks using approximated backpropagation with analog synaptic devices. *Frontiers in Neuroscience*, Vol. 14, p. 423, 2020.

[76] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34, No. 10, pp. 1537–1557, 2015.

[77] J. M. Cruz-Albrecht, M. W. Yung, and N. Srinivasa. Energy-efficient neuron, synapse and STDP integrated circuits. *IEEE Transactions on Biomedical Circuits and Systems*, Vol. 6, No. 3, pp. 246–256, 2012.

[78] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4, 2011.

[79] Deliang Fan, Yong Shim, Anand Raghunathan, and Kaushik Roy. STT-SNN: A Spin-Transfer-Torque Based Soft-Limiting Non-Linear Neuron for Low-Power Artificial Neural Networks. *IEEE Transactions on Nanotechnology*, Vol. 14, No. 6, pp. 1013–1023, November 2015.

[80] Qi Deng, Yi Cheng, and Guanghui Lan. Optimal Adaptive and Accelerated Stochastic Gradient Descent. *arXiv:1810.00553 [cs, stat]*, October 2018.

[81] Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham M. Kakade. On the insufficiency of existing momentum schemes for Stochastic Optimization. *arXiv:1803.05591 [cs, math, stat]*, July 2018.

[82] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients. *arXiv:2010.07468 [cs, stat]*, December 2020.

[83] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive Gradient Methods with Dynamic Bound of Learning Rate. *arXiv:1902.09843 [cs, stat]*, February 2019.

[84] Jianbang Ding, Xuancheng Ren, Ruixuan Luo, and Xu Sun. An Adaptive and Momental Bound Method for Stochastic Learning. *arXiv:1910.12249 [cs, stat]*, October 2019.

[85] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive Learning Rates with Sublinear Memory Cost. *arXiv:1804.04235 [cs, stat]*, April 2018.

[86] Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W. Mahoney. ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning. *arXiv:2006.00719 [cs, math, stat]*, November 2020.

[87] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. AdamP: Slowing Down the Slowdown for Momentum Optimizers on Scale-invariant Weights. *arXiv:2006.08217 [cs, stat]*, October 2020.

[88] Xuezhe Ma. Apollo: An Adaptive Parameter-wise Diagonal Quasi-Newton Method for Nonconvex Stochastic Optimization. *arXiv:2009.13586 [cs, stat]*, November 2020.

[89] Shiv Ram Dubey, Soumendu Chakraborty, Swalpa Kumar Roy, Snehasis Mukherjee, Satish Kumar Singh, and Bidyut Baran Chaudhuri. diffGrad: An Optimization Method for Convolutional Neural Networks. *arXiv:1909.11015 [cs, math]*, March 2020.

[90] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. *arXiv:1904.00962 [cs, stat]*, January 2020.

[91] Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead Optimizer: K steps forward, 1 step back. *arXiv:1907.08610 [cs, stat]*, December 2019.

[92] Boris Ginsburg, Patrice Castonguay, Oleksii Hrinchuk, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Huyen Nguyen, Yang Zhang, and Jonathan M. Cohen. Stochastic Gradient Methods with Layer-wise Adaptive Moments for Training of Deep Networks. *arXiv:1905.11286 [cs, stat]*, February 2020.

[93] Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and Adam for deep learning. *arXiv:1810.06801 [cs, stat]*, May 2019.

[94] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv:1908.03265 [cs, stat]*, April 2020.

[95] Qianqian Tong, Guannan Liang, and Jinbo Bi. Calibrating the Adaptive Learning Rate to Improve Convergence of ADAM. *arXiv:1908.00700 [cs, math, stat]*, September 2019.

[96] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv:1608.03983 [cs, math]*, May 2017.

[97] Nitish Shirish Keskar and Richard Socher. Improving Generalization Performance by Switching from Adam to SGD. *arXiv:1712.07628 [cs, math]*, December 2017.

[98] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned Stochastic Tensor Optimization. *arXiv:1802.09568 [cs, math, stat]*, March 2018.

[99] Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures. *Frontiers in Neuroscience*, Vol. 14, , February 2020.

[100] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 29, No. 7, pp. 3227–3235, July 2018.

[101] Yunzhe Hao, Xuhui Huang, Meng Dong, and Bo Xu. A biologically plausible supervised learning method for spiking neural networks using the symmetric STDP rule. *Neural Networks*, Vol. 121, pp. 387–395, January 2020.

[102] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*, September 2017.

[103] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: An extension of MNIST to handwritten letters. *arXiv:1702.05373 [cs]*, March 2017.

[104] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, Vol. 115, No. 3, pp. 211–252, December 2015.